# 4



# Spreadsheet Proğramının Beyninin Kodlanması

Sabik iki böümde Spreadsheet proğramının kullanıcı arabiriminin nasıl oluşturulduğunu gördük. Bu bölümde Elektronıik çizelge proğramının beyni diye nitelendirebileceğimiz kısmını kodlayacağız. Bu bölümde öğreneceklerimizin başında, dosya yükleyüp kaydetmek, bellekte data muhafaza etmek ve pano operasyonları (kes ve yapıştır gibi) ile QTable sınıfına elektronik çizelge proğramında kullanılacak olan formülleri anlama yeteneğinin kazandırılması geliyor.

## *Merkezi Alet*

QMainWindow un merkezi kısmını herhangi bir alet işgal edebilir. İşte muhtemel bir kaç metod:

1. **Standart Qt aleti kullan.**
   QTable veya QTextEdit gibi standart Qt aleti merkezi alet olarak kullanılabilir. Bu durumda, proğramın dosya yükleme ve kaydetme gibi faliyetleri başka bir yerde ifa edilmelidir (mesela bir QMainWindow alt sınıfında ).
2. **Özel bir alet kullan.**
   Özel amaçlı proğramlar genellikle hususi tasarlanmış aletlere ihtiyaç duyarlar. Mesela, bir simge muharriri (IconEditor) merkezi aleti olarak doğaş olarak IconEditor aleti kullanır. Beşinci bölümde Qt altında hususi aletlerin nasıl yapılabileceğini işleyeceğiz.
3. **Bir dizim (layout) ile birlikte adi bir QWidget kuullan.**
   Bazan bir ptoğramın merkezi bölümü bürden fazla alet tarafından istila edilmiş olabilir. Bütün bu altlerin atası olaran bir QWidget ile çocuk aletlerin mahalleri ve ebatlarını tanzim etmek için bir dizim kullanmak suretiyle bu geröekleştirilebilir.
4. **Ayırıcı (splitter) kullan.**
   Müteaddid aletleri bir arada kullanmanın yollarından biriside bir ayırıcı sınıfı olan Qsplitter kullanmaktır. QSplitter çocuklarını ya QHBox gibi yatay olarak yan yana veya QHBox gibi düşey olarak çocuklar arasında kullanıcının yer değşitirebileceği bir ayırıcı olacak şekilde tertib eder ki buda kullanıcıya aletlerin ebatlarını tebdil etmesini mümkün kılar. Ayırıcılar her tür alet ihtiva ettiklri gibi diğer ayırıcılar dahi içerebilirler.

5. **MDI kullan.**
   Eğer proğram MDI kullanıyorsa bu durumda merkezi alannını QWorkspace aleti (iş alanı) kaplar, ve MDI .öerisindeki her bir pencere QWorkspace aletinin çocuğuduralr.

Dizim mekanizmaları (layouts), ayırıcılar (splitters), ve MDI iş alanları diğer Qt standart aletleri ile kullanılabilecekleri gibi husui aletler ile de kullanılabilirler. Altıncı bölümde esnafı sabıka tafsilen ele alınacaktır.

Spreadsheet proğramında QTable ın bir alt sınıfı merkezi alet olarak kullanıldı. QTable bir elektronik çizelge proğramında ihtiyaç duyulan çoğu işlevi tedarik etmekle birlikte =A1+A2+A3 gibi formüller ile kes yapıştır panosu operasyonlarını desteklememektedir. Proğramımızca şhtiyaç duyulan bu istidadları QTable ın alt sınıf olan Spreadsheet sınıfına kodlayacağız.

## *QTable dan Altsınıf Oluşturulması*

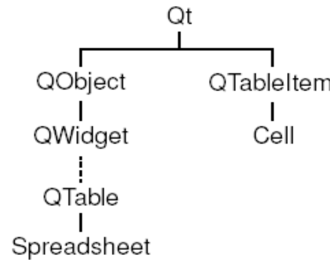
Başlık dosyasından başlayarak Spreadsheet aletini kodlayacağız:

```
#ifndef SPREADSHEET_H
#define SPREADSHEET_H

#include <qstringlist.h>
#include <qtable.h> class Cell;

class SpreadsheetCompare;
```

Başlık dosyasında önce Cell ve SpreadsheetCompare sınıflarının ön tanımlarını yapıyoruz.




**Şekil 4.1**: Spreadsheet ve Cell (hücre) sınıflarının soy ağacı.

QTable hücrelerinin, metin (text) ve hiza (alignment) gibi vasıfları QtableItem içerisinde muhafaza edilmektedirler. QTable sınıfının aksine QTableItem bir alet sınıfı olmayıp bilakis saf bir veri veya data sınıfıdr. Cell sınıfı ise QTableItem ın bir alt sınıfıdır. Standart QTableItem vasıflarına ilaveten Cell sınıfı hücreye ait formülü muhafaza eder.

Bu bölümğn son kısmında Cell sınıfının kodunu yazarken bu sınıf hakkında detaylı bilgi verilecektir.

```
class Spreadsheet : public QTable
{
```

```
        Q_OBJECT
    public:
        Spreadsheet(QWidget *parent = 0, const char *name = 0);
        void clear[1]();
        QString currentLocation[2]() const;
        QString currentFormula[3]() const;
        bool autoRecalculate() const { return autoRecalc; }
        bool readFile(const QString &fileName);
        bool writeFile(const QString &fileName);
        QTableSelection selection();
        void sort(const SpreadsheetCompare &compare);
```

Spreadsheet sınıfı QTable sınıfının varisidir (alt sınıfıdır). QTable dan alt sınıf oluşturulması QDialog veya QmainWindow dan alt sınıf oluşturulması gibidir.

Üçüncü bölümde, MainWindow oluşturuken Spreadsheet sınıfının umumi (public) fonksiyonlarından bir hayli yararlandık. Mesela, MainWindow:: newFile() içerisinden clear() fonksiyonunu çağırmak suretiyle elektronik çizelgeyi yeni dosya için hazır hale getirdik (reset). Yine QTable dan miras kalan setCurrentCell() gibi setShowGrid() fonksiyonları kullandık.

```
    public slots:
      void cut();
      void copy();
      void paste();
      void del();
      void selectRow();
      void selectColumn();
      void selectAll();
      void recalculate();
      void setAutoRecalculate(bool on);
      void findNext(const QString &str, bool caseSensitive);
      void findPrev(const QString &str, bool caseSensitive);
    signals:
      void modified();
```

Spreadsheet Düzenle, Aletler ve Seçenekler menülerinin işlevlerini yerine getiren bir çok dilim (slot) tedarik etmektedir.

```
    protected:
      QWidget *createEditor(int row,int col,bool initFromCell) const;
      void endEdit(int row,int col,bool accepted,bool wasReplacing);
```

Spreadsheet, QTable sınıfının iki fonksiyonunu yeniden tanımlar, kı bunlar QTable tarafından, kullanıcı hücre içindeki değeri değiştirmeye başlayınca, çağrılırlar. Elektronik çizelgenin formülleri anlayabilmesi için bu iki fonksiyonu yeniden tanımlamamız gerekli.

```
    private:
```

[1] sil
[2] aktif yer
[3] aktif formül

```
    enum { MagicNumber[4] =0x7F51C882, NumRows[5] =999, NumCols[6] =26 };

    Cell *cell(int row, int col) const;
    void setFormula(int row, int col, const QString &formula);
    QString formula(int row, int col) const;
    void somethingChanged[7]();
    bool autoRecalc;
};
```

Sınıfın hususi (private) kısmında üç tane sabit değişken (constants), dört fonksiyon ve bir değişken tanımlıyoruz.

```
class SpreadsheetCompare
{
      public:
            bool operator()(const QStringList &row1,
                            const QStringList &row2) const;
            enum { NumKeys = 3 };
            int keys[NumKeys];
            bool ascending[NumKeys];
};

#endif
```

Başlık dosyası SpreadsheetCompare sınıfının tanımıyla sona erdi. Bunu Spreadsheet::sort() sınıfını ele alırken açıklayacağız. Şimdi başlık dosyasında tanımlanan fonksiyon ve sınıflara oşlevlik verecek kodu yazacağız ve ger bir fonksiyonu açıklayacapız:

```
#include <qapplication.h>
#include <qclipboard.h>
#include <qdatastream.h>
#include <qfile.h>
#include <qlineedit.h>
#include <qmessagebox.h>
#include <qregexp.h>
#include <qvariant.h>

#include <algorithm>
#include <vector>
using namespace std;

#include "cell.h"
#include "spreadsheet.h"
```

Proğramda kullanılacak olan Qt sınıflarının başlık dosyalarını dahil etmekle birlikte standart C++ ta mevcut olan <algorithm> ve <vector> başlık dosyalşarınıda dahil ediyoruz. using namespace komutu std mıntıkasındaki (namespace) bütün sembolleri küresel yada global mıntıkaya ıthal eder böylece std mıntıkasındaki mevcut herşeye std:: önekini kullanmaksızın ulaşabiliriz. Bu demek oluyorki std::stable_sort() ve std::vector<T> terine

4 sihirli rakam
5 satır sayısı
6 sütun sayısı
7 birşey değişti

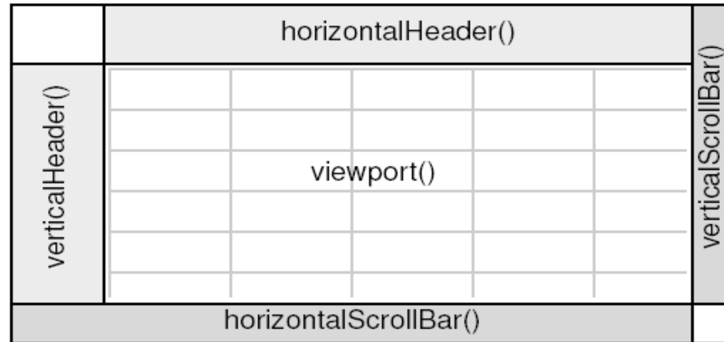
stable_sort() ve vector<T> kullanarak bunlara ulaşabiliriz.

```
Spreadsheet::Spreadsheet(QWidget *parent, const char *name)
      : QTable(parent, name)
{
      autoRecalc = true;
      setSelectionMode(Single)[8];
      clear();
}
```

Yapıcıda (constructor) QTable seçme modune Single (tek) olarak belirledik. Buda elektronik çizelge proğramında her defasında ancak bir dikdörtgenlik alan seçilebilir.

```
void Spreadsheet::clear()
{
      setNumRows(0);
      setNumCols(0);
      setNumRows(NumRows)[9];
      setNumCols(NumCols)[10];
      for (int i = 0; i < NumCols; i++)
            horizontalHeader()->setLabel(i, QChar( A  + i));
      setCurrentCell(0, 0);
}
```

clear() fonksiyonu Spreadsheet yapıcısı (constructor) tarafından elektronik çizelgeyi hazırlamak için çağrılır. Bu fonksiyon, yukarıdada belirtildiği gibi, MainWindow::newFile() tarafındanda çağrılır. Biz ünce elektronik öşzelgeyi 0 × 0 boyutuna getirerek bir bakıma onun muhteviyatını siliyoruz ve daha sonra boyutunu NumCols × NumRows (26 × 999) olarak deüiştiriyoruz. Sütun etiketlerini A , B , &, Z olarak değiştiriyoruz (bunun sebebi QTable da sütunlar 1 , 2 , &, 26 şekline numaralandırılmalarıdır ) ve imleci A1 hücresine yerleştiriyoruz.




**Şekil 4.2**: QTable bileşeninin ihtiva ettiği aletler.

QTable çok sayıda çocuk bileşeninden ibarettir. Üstte yatay bir QHeader (başlık), solda düşey bir QHeader sağda düşey QScrollBar (kaydırma çubuğu) ve altta yatay bir QScrollBar mevcuttur. Merkezi alan vievport diye adlandırılan bir bileşim tarafından işgal

8 seçmeModunuBelirle(tek)

9 satırSayısınıTesbitEt(satırSayısı)

10 sütunSayısınıTesbitEt(sütunSayısı)

edilmektedir, ki bu alet üzerine QTable hücrelerini çizer. Muhtelif çocuk aletlere QTable ve onun üst sınıfı olan QScrollView içerisinde mevcut fonksiyonlar vasıtasıyla ulaşılabilir. Mesela, clear() fonksiyonu içerisinde, tablonun üst başlığına (QHeader ) QTable:: horizontalHeader() vasıtasıyla ulaşıyoruz.



## Dataları Ferd (Item) Olarak Bellekte Saklamak

Elektronik çşzelge proğramında, boş olmayan her bir hücre hafızada ferdi QTableItem nesnesi olarak saklanır. Bu şekilde dataların bellekte tutulması QTable sınıfına mahsus değildir; Qt nin QIconView, QListBox, ve QListView sınıflarıda fertlerle c(QIconViewItems, QListBoxItems, and QlistViewItems) çalışırlar.

Qt nin bu sınıflarını bu kitapta içeren yada ihtiva eden manasında muhtevi sınıfları diye tabir edeceğiz ki onlar hiç bir değişikliğe gerek kalmadan data tutmak için kullanılabilirler. Mesela, QTableItem hali hazırda bir metin (string), bir pixmap ve birde QTable için bir müşir (pointer) ihtiva eder. QTable sınıfından alt sınıf oluğturmak suretiyle bir kaç data daha ekleyebilir ve bu data üzerinde iş yapacak bir kaç hayali (virtual) fonksiyon ekleyebiliriz.

Çok sayıda kütüphaneler (toolkits) ferd sınıflarında hususi dataların tutulabilmesi için umumi müşir tedarik ederle. Qt, gereksiz yere her bir ferde müşir ilave etmek yerine, proğramcının ferd sınıfından bir alt sınıf oluşturarak gerekli dataları orada tutmasını veya varsa mevcut dataya bir müşir eklemesini mümkün kılar. Eğer umumi müşire (void pointer) ihtiyaç duyulursa bu durumda ferd sınıfından bir alt sınıf oluşturulur ve umumi müşir bu sınıfa üye değişken olarak eklenir.

QTable sınıfının alte seviye fonksüyonları ola paintCell() ve clearCell() yeniden tanımlamak suretiyle ferd makanizmasının kullanımasına gerek kalmayabilir. Hücrelerde görüntülenecek olan data hafızada başka bir data yapısında mevcut ise ferd mekanizmasının kullanılmaması müreccehhdir taki aynı datanın hafızada birden fazla yerde tutulması önlenmiş olsun. Ayrıntıları Qt Quarterly da yer alan A Model/View Table for Large Datasets pasajında bulabilirsiniz kı bu magazini ağda şu URL de bulabilirsiniz: http://doc.trolltech.com/qq/qq07-big-tables.html.

Qt 4 ün Qt 3 ten data tutma konusunda daha esnek olması bekleniyor. Qt 4, ferdleri

QScrollView sınıfı doğal olarak cok miktarda datayı teşhir edecek olan sınıfların üst sınıfıdır. O kaydırılabilir bir viewport (bkz. Şekil 4.2) ile iki tanede kaydırma çubuğu tedarik eder ki bu çubuklar arzu edildiğinde saklanabilirler. Altıncı bölümde bu sınıfı detaylı olarak ele alacağız.

```
Cell *Spreadsheet::cell(int row, int col) const
{
      return (Cell *)item(row, col);
}
```

Hususi cell() fonksiyonu belirli bir satır ve sütuna ait olan Cell nesnesine verir. Bu hemen hemen Qtable::item() fonksiyonu ile aynıdır, ancak Cell müşiri yerine QTableItem müşiri verir.

```
QString Spreadsheet::formula(int row, int col) const
{
      Cell *c = cell(row, col);
      if (c)
            return c->formula();
      else
            return "";
}
```

Hususi formula() fonksiyonu her bir hücreye ait formülü verir. Eğer cell() bir boş müşir verirse bu hücrenin boş olması anlamına gelir böylece boş bir metin geri gönderiyoruz.

```
void Spreadsheet::setFormula(int row, int col,
                              const QString &formula)
{
      Cell *c = cell(row, col);
      if (c)
      {
            c->setFormula(formula);
            updateCell(row, col)[11];
      }
      else
      {
            setItem(row, col, new Cell(this, formula));
      }
}
```

Hususi setFormula() fonksiyonu sıra ve sütun numarası verilen hücrenin formülünü tesbit eder. Eğer bu hücrenin hali hazırda Cell nesnesi mevcut ise onu kullanıyoruz ve updateCell() fonksiyonunu çağırark QTable a hücre şayet ekranda görünen bir pozisyonda ise onu guncelleştirmesini istiyoruz. Aksi takdirde bir Cell nesnesi meydana getiriyoruz ve QTable::setItem() fonksiyonunu çağırarak onu tabloya yerleştirip ekranda göstermesini sağlarız. Cell nesnesini daha sonra silinmesi hususunda kafa yormamıza gerek yok çunkü QTable bu nesneyi sahipleşir ve gerektiği vakitte onu siler.

```
QString Spreadsheet::currentLocation() const
```

---

[11] hücreyiGüncelleştir(sıra, sütun)

```
{
      return QChar( A  + currentColumn()) +
                  QString::number(currentRow() + 1);
}
```

currentLocation() fonksiyonu aktif hücre numarasını sutun harfı ve ardından satır numarası olarak verir. Bu fonksiyon MainWindow:: updateCellIndicators() tarafından aktif hücre numarasının durum çubuğunda gösterilmesi için istimal edilir.

```
QString Spreadsheet::currentFormula() const
{
      return formula(currentRow(), currentColumn());
}
```

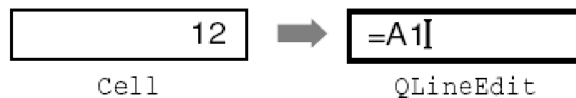
currentFormula() fonksiyonu aktif hücrenin formülünü verir. Bu fonksiyon MainWindow:: updateCellIndicators() tarafından çağrılır.

```
QWidget *Spreadsheet::createEditor(int row, int col,
                                    bool initFromCell) const
{
      QLineEdit *lineEdit = new QLineEdit(viewport());
      lineEdit->setFrame(false);
      if (initFromCell)
            lineEdit->setText(formula(row, col));
      return lineEdit;
}
```

createEditor() fonksiyonu Spreadsheet sınıfına QTable dan miras kalmakla birlikte burada yeniden tanımlanmaktadır.  Bu fonksiyon kullanıcı hücreye tıklayarak, F2 tuşuna basarak veya basitçe yazmaya başlamak suretiyle değiştirmeye kalktıpında şağrılır. Bunu görevi bir muharrir birleşim oluşturmaktırki bu alet hücrenin üzerinde görüntülenir. Kullanıcı içeriğini değiştirmek için hücreye tıklarsa veya F2 tuşuna basarsa, initFromCell değişkeni müsbet (true) olur ve muharrir aktif hücrenin işeriğini ihtiva eder bir şekilde başlatılır. Kullanıcı hücreye tıklamaksızın veya F2 tuğuna basmaksızın yazmaya başlarsa bu durumda hücrenin içeriği ihmal edilir.

Bu fonksiyonun normalde bir QLineEdit oluşturur be initFromCell değişkeni müsbet ise bu muharrire hücrenin içeriğini kopyalar. Biz bu fonksiyonu yeniden tanımladık taki hücrenin içeriği yerine onun formülünü göstersin.

QTable ın viewport unun çocuğu olan bir QLineEdit oluşturuyoruz. QTable bunun boyutlarını değiştirilmekte olan hücre ebatlarına göre ayarlar ve sözkonusu hücrenin üzerine yerleştirir. QTable, ihtiyaç duyulmadığı takdirde  QLineEdit i siler.




**Şekil 4.3**: Hücre ve değşitirme esnasında üzerine yerleştirilen QlineEdit.

Çoğu zaman formül ve metin aynıdır, mesela Merhaba formülnün sonucu Merhaba metnidir.

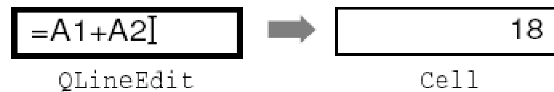
Yani bir hücreye Merhaba yazıp Enter tuşuna basıldığında o hücre Merhaba metnini görüntüler. Bunun istisnaları şunlardır:

- Eğer formül bir rakam ise o şekilde algılanır. Mesela formül 1,50 ise bu formülün sonucu bir doğal sayı olan 1,5 dir. Hücrede bu sayı sağa yanaşmış bir şekilde gösterilir.
- Eğer formül “'” işareti ile başlarsabu durumda formül metin olarak algılanır. Mesela '12345' formülünün değeri 12345 metnidir.
- Formül eşit işaretiyle ( = ) ile başlarsa bu durumda formül aritmetik ibare olarak algılanır. Mesela, A1 hücresü 12 ve A2 hücresi 6 içeriyorsa, =A1+A2 formülünün sonucu 18 dir.

Formüllerin değerlendirilmesi vazifesi Cell sınıfına aittir. Şu aşamada akılda bulundurulması greken husus hücrede gösterilen metnin formülün kendisi değil bilakis formülün değerlendirilmesi sonucu elde edilen değer olmasıdır.

```
void Spreadsheet::endEdit(int row, int col,
                  bool accepted, bool wasReplacing)
{
      QLineEdit *lineEdit = (QLineEdit *)cellWidget(row, col);
      if (!lineEdit) return;
            QString oldFormula = formula(row, col);
            QString newFormula = lineEdit->text();
            QTable::endEdit(row, col, false, wasReplacing);
      if (accepted && newFormula != oldFormula)
      {
            setFormula(row, col, newFormula);
            somethingChanged();
      }
}
```

QTable sınıfının endEdit() fonksiyonu burada yeniden tanımlandı. Bu fonksiyon kullanıcı hücreyi değiştirme işlemini bitirdiğinde iağrılır, ki bu elektroniş öizelge üzerinede başka bir alana tıklamakla veya Enter tuşuna (yapılan değişikliği tasdik manasında) veya Esc tuşuna (yapılan değişikliği red manasında) basarakla yapılır. Bu fonksiyonun görevi muharririn içeriğini (QLineEdit), yapılan değişikliğin tasdik edilmesi durumunda, hücreye geri göndermektir. Muharrire Qtable::cellWidget() vasıtası ile ulaşılabilir. Bu fonksiyonun getirdiği aleti rahatlıkla QLineEdit e çevirebiliriz çünkü createEditor() içerisinde ouşturulan alet daima QLineEdit dir.




**Şekil 4.4**: QlineEdit in içeriğinin hücreye kopyalanması.

Fonksiyonun ortasında QTable sınıfının endEdit() fonksiyonunu çağırıyoruz çünkü QTable değiştirme işlemini ne zaman bittiğini bilmesi lazım. endEdit() fonksiyonunun üçüncü argümanı olarak menfi (false) kullanıyoruz taki tablonun ferdini değiştirmesin çünkü bu değişikliği biz yapacağız. Eğer yeni formül eskisinden farklı ise setFormula() fonksiyonunu çağırmak suretiyle Cell nesnesini tadil edip sonrada somethingChanged() fonksiyonu

çağrılır.

```
void Spreadsheet::somethingChanged()
{
      if (autoRecalc)
            recalculate();
      emit modified();
}
```

Hususi bir fonksiyon olan somethingChanged(), elektronik çizelegeni tamamını,seşeneği müsbete ayarlanmış ise, yeniden hesaplar ve modified() sinyalini yayınlar.

## *Yükleme ve Kaydetme*

Şimdi Spreadsheet proğramının dosya yükleme ve kaydetme işlevini hususi bir binary format ile yapabilmesi için gerekli kodu yazacağız. Bu işlemi QFile ve QdataStream sınıflarını kullanarak işletim sisteminden bapımsız binary I/O olarak gerçekleştireceğiz. Spreadsheet dosyasının kaydedilmesi ile işe başlayacağız:

```
bool Spreadsheet::writeFile(const QString &fileName)
{
      QFile file(fileName);
      if (!file.open(IO_WriteOnly))
      {
      QMessageBox::warning(this, trUtf8("Spreadsheet"),
                        trUtf8("%1 dosyas  kaydedilemiyor:\n%2.")
                        .arg(file.name())
                        .arg(file.errorString()));
      return false;
      }
      QDataStream out(&file);
      out.setVersion(5);
      out << (Q_UINT32)MagicNumber;
      QApplication::setOverrideCursor(waitCursor);
      for (int row = 0; row < NumRows; ++row)
      {
            for (int col = 0; col < NumCols; ++col)
            {
                  QString str = formula(row, col);
                  if (!str.isEmpty())
                        out << (Q_UINT16)row
                            << (Q_UINT16)col
                            << str;
            }
      }
      QApplication::restoreOverrideCursor();
      return true;
}
```

writeFile() fonksiyonu MainWindow::saveFile() tarafından dosyayı kaydetmek için çağrılır. Kaydetme işlemi başarıyla tamamlandı ise müsbet aksi takdirde menfi geri getirir.

Verilen dosya ismini kullanarak bir QFile nesnesi oluşturup open() fonksiyonunu çağırarak

dosyayı açıyoruz. Aynı zamanda bir QDataStream nesnesi oluşturuyoruz ki bu nesne QFile üzerinde çalışmaktadır ve datanın kaydedilmesi için kullanılır. Datayı yazmadan önce proğramın imleçini bekle moduna alıp (genelde kum saati şekline görünür) datayı yazdıktan sonra imleçi normal haline çeviriyoruz. Fonksiyonun sonunda dosya QFile imha edicisi (destructor) tarafından otomotik olarak kapatılır.

QDataStream temel C++ datat türleri ile öalışabildiği gibi çok sayıda Qt datası ilede çalışabilir. QDataStream ın nehvi standart <iostream> sınıflarına baenzer. Mesela ,

out << x << y << z;

x,y, ve z değişkenlerini bir stream e yazar ve

fin >> x >> y >> z;

bu değişkenleri bir stream den okur.

Temel C++ tipleri olan char, short, int, long, ve long long farklı platformlarda farklı büyüklüğe sahip olabilir, bu bakımdan bunları Q_INT8, Q_UINT8, Q_INT16, Q_UINT16, Q_INT32, Q_UINT32, Q_INT64, ve Q_UINT64 çevirmek suretiyle büyüklüklrinin bütün platformlarda aynı olmasını garanti edebiliriz.

QDataStream oldukça elverişlidir. QFile ile kullanıldığı gibi QBuffer, Qsocket ve QsocketDevice ile de kullanılabilir. Benzer şekilde QFile, QdataStream yerine QTextStream ile hatta raw (çiğ) data ile dahi kullanılabilir. Onuncu bölümde bu sınıflar tafsilen ele alınacaklardır.

Spreadsheet proğramının dosya formatı gayet basittir. Bir Spreadsheet dosyasi 32-bit bir yayı ile başlar ki bu sayı (MagicNumber diye 0x7F51C882 olarak spreadsheet.h dosyasında tanımlandı) dosyanın formatını gösterir. Bunun ardından her bir hücrenin sаır, sütun numarası ile formülünü içeren bloklar gelir. Diskte fazla yer kaplamamk için boş hücreleri katdetmiyoruz.




**Şekil 4.5**: Spreadsheet proğramının dosya formatı.

Data türlerinin tam anlamı ile ne şekilde kaydedildiği QdataStream tarafından belirlenir. Mesela, bir Q_UINT16 big-endian sırasında iki byte ile temsil edilir, ve bir QString uzunluğunu mütaakip Unicode karekterleri tarafından temsil edilir.

Qt türlerini binary formatında temsil edilme şekli Qt 1.0 dan bu tana bir hayli değişti. Muhtemelen istikbaldede değişmeye devam edecektir bunu sebebei ise mevcut türlerin değişmeşisine izin vermek ve başka tiplerin eklenmesine izin vermektir. By default, QDataStream en yeni binary formatı (Qt 3.2 altında, versiyon 5), kullanmaktadır, ancak eski versiyonları okumakta mümkündür. Spreadsheet proğramının, Qt nin daha yeni versiyonları ile derlendiğinde, uyarlılık problemlerini engellemek için QDataStream ın

daima 5 versiyonunu kullanmasını sağlayacağız.

```
bool Spreadsheet::readFile(const QString &fileName)
{
      QFile file(fileName);
      if (!file.open(IO_ReadOnly))
      {
            QMessageBox::warning(this, trUtf8("Spreadsheet"),
                        trUtf8("Dosya okunamad  %1:\n%2.")
                        .arg(file.name())
                        .arg(file.errorString()));
      return false;
      }
      QDataStream in(&file);
      in.setVersion(5);
      Q_UINT32 magic;
      in >> magic;
      if (magic != MagicNumber)
      {
            QMessageBox::warning(this, truTF8("Spreadsheet"),
                        truTF8("Bu dosya geçerli bir "
                        "Spreadsheet dosyas  de ildir"));
            return false;
      }
      clear();
      Q_UINT16 row;
      Q_UINT16 col;
      QString str;
      QApplication::setOverrideCursor(waitCursor);
      while (!in.atEnd())
      {
            in >> row >> col >> str;
            setFormula(row, col, str);
      }
      QApplication::restoreOverrideCursor();
      return true;
}
```

readFile() fonksiyonu writeFile() fonksiyonuna çok benzer. QFile kullanarak dosyayı okuyoruz andcak bu defa IO_WriteOnly yerine IO_ReadOnly seçeneğini kullanıyoruz. Sonra QDataStream versiyonunu 5 olarak ayarlıyoruz. Okurken kullanılan format yazarken kullanılan formatın aynısı olmalıdır.

Eğer dosya sihirli numara ile başlıyorsa clear() fonksiyonunu çağırarak bütün hücreleri boşaltıyoruz ve datayı okuyoruz. Dosyada yer almayan hücreleri temizlemek için clear() fonksiyonunu kullanmak eşzemdir.

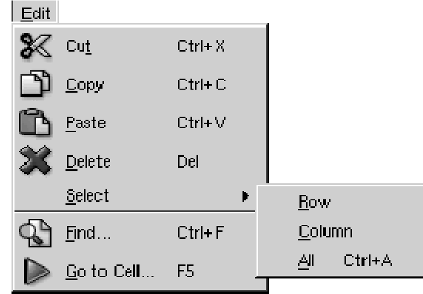
## *Düzenle Menüsü*

Şimdi sıra Düzenle menüsü için gerekli olan dilimleri proğramlamaya geldi.

```
void Spreadsheet::cut()
{
```

```
        copy();
        del();
}
```

cut() dilimi Düzenle|Kes menüsüne tekabul eder. Kes komutu aslında ünce kopyala ve sonra sil komutundan ibaret olduğu işin kodlanması gayet basittir.




**Şekil 4.6**: Spreadsheet proğramının Düzenle menüsü.

```
void Spreadsheet::copy()
{
        QTableSelection sel = selection();
        QString str;
        for (int i = 0; i < sel.numRows(); ++i)
        {
                if (i > 0) str += "\n";
                for (int j = 0; j < sel.numCols(); ++j)
                {
                if (j > 0)
                        str += "\t";
                str += formula(sel.topRow() + i, sel.leftCol() + j);
                }
        }
        QApplication::clipboard()->setText(str);
}
```

copy() dilimi Düzenle|Kopyala menüsüne tekabul eder. Seçilmiş alan içerisindeki bütün hücreler üzerinde çalışır. Seçilmiş olan her bir hücrenin formülü bir QString te tutlur ve sıralar yeni satır karakteri ile ve sütunlar ise tab ile birbirlerinden ayrılırlar.

| | C | D | E |
|---|---|---|---|
| 2 | Red | Green | Blue |
| 3 | Cyan | Magenta | Yellow |



"Red\t Green\t Blue\n Cyan\t Magenta\t Yellow"

**Şekil 4.7**: Seçilmiş bir kısmın panoya kopyalanması.

Qt altında işletim sisteminin panosuna QApplication::clipboard() sakin (static) fonksiyonu vasıtası ile ulaşılabilir. Qclipboard::setText() fonksiyonunu çağırmak suretiyle metni panoda hem bu proğramın hemde sade metin ile çalışabilen diğer bütün proğramların emrine amade edebiliriz. Kullanmış olduğumuz, tab ve yeni satır işaretlerini içeren format, Microsoft Excel başta olmak üzere bir çok diğer proğramlar tarafından anlaşılmaktadır.

```
QTableSelection Spreadsheet::selection()
{
      if (QTable::selection(0).isEmpty())
      return QTableSelection(currentRow(), currentColumn(),
                             currentRow(), currentColumn());
      return QTable::selection(0);
}
```

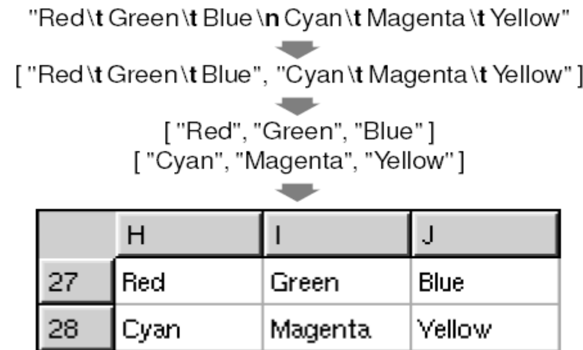
Hususü selection() fonksiyonu hali hazırda seçilmiş olan alanı verir. Bu fonksiyon Qtable::selection() fonksiyonuna itimad eder. Qtable::selection() fonksiyonu seçilmiş alanları numaralandırmaktadır. Biz yanlızca bir alanın seçilmesine izin verdiğimiz için 0 numaralı alanı taleb ediyoruz. Hiç bir hücre veya hücreler gurubunun seçilmemiş olmasıda tabiiki mümkün. Bunu sebebi QTable ın aktif hücreyi seçilmiş ittihaz etmez. Bu davranış aslında makul olmakla birlikte bizim için en müsait bir seçenek olmadığından biz burada selection() fonksiyonunu yeniden tanımladık ve o artık varsa seçilmiş olan alanı aksi takdirde aktif hücreyi verir.

```
void Spreadsheet::paste()
{
      QTableSelection sel = selection();
      QString str = QApplication::clipboard()->text();
      QStringList rows = QStringList::split("\n", str, true);
      int numRows = rows.size();
      int numCols = rows.first().contains("\t") + 1;

      if (sel.numRows() * sel.numCols() != 1
            && (sel.numRows() != numRows
            || sel.numCols() != numCols))
      {
            QMessageBox::information(this, tr("Spreadsheet"),
               trUtf8("Kopyalanan ve yap t r lacak olan alanlar "
               "farkl  oldu u  için yap t r ma"
               "i lemi  gerçekle tirilemedi."));
            return;
      }
      for (int i = 0; i < numRows; ++i)
      {
            QStringList cols =
                  QStringList::split("\t", rows[i], true);
            for (int j = 0; j < numCols; ++j)
            {
                  int row = sel.topRow() + i;
                  int col = sel.leftCol() + j;
```

```
                if (row < NumRows && col < NumCols)
                    setFormula(row, col, cols[j]);
                }
            }
            somethingChanged();
        }
```

paste() dilimi  Düzenle|Yapıştır dilimine tekabul eder. Panodaki metni alıp sakin QstringList::split() fonksiyonunu çağırmak suretiyle metni QStringList tğrğnde bir listeye çeviriyoruz. Her satır QStringList listesinin bir elamanı olur. Daha sonra kopya yapılacak alanın ebatlarını belirliyoruz. Satır sayısı QStringList listesindeki elemean sayısına eşittir. Sütun sayısı ise ilk satırdaki tab sayısı artı birdir. Sadece bir hücre seçilmiş ise biz onu yapıştırma alanının sol üst kmşesi ittihaz ediyoruz. Aksi taktirde hali hazırdaki seçilmiş alanı yapıştırma alanı olarak algılıyoruz. Yapıştırma işlemini ifa etmek için her bir satırı, tab işaretini ayırıcı olarak kullanmak suretiyle QStringList::split() fonksiyonun çağırıp, hücrelere bölüyoruz.  Şekil 4.8 yapıştırma işleminin ihtiva ettiği adımları izhar etmektedir.




**Şekil 4.8**: Panodaki metnin elektronik çizelgeye yapıştırılması.

```
void Spreadsheet::del()
{
    QTableSelection sel = selection();
    for (int i = 0; i < sel.numRows(); ++i)
    {
        for (int j = 0; j < sel.numCols(); ++j)
            delete cell(sel.topRow()+i,sel.leftCol()+j);
    }
    clearSelection();
}
```

del() dilimi Düzenle|Sil menüsüne tekabul eder. Seçilmiş alanı silmek için delete komutunu kullanarak her bir Cell nesnesini silmek kafidir. QTable ihtiva ettiği QTableItems ferdlerinin silinmiş olduğunun farkına varır ve kendisini ekranda yeniden gösterir. cell() fonksiyonunu silinmiş olan bir hücre için çağırırsak o boş bir müşir verir.

```
void Spreadsheet::selectRow()
{
    clearSelection();
    QTable::selectRow(currentRow());
}
```

```
void Spreadsheet::selectColumn()
{
      clearSelection();
      QTable::selectColumn(currentColumn());
}
void Spreadsheet::selectAll()
{
      clearSelection();
      selectCells(0, 0, NumRows - 1, NumCols - 1);
}
```

selectRow(), selectColumn(), ve selectAll() fonksiyonları sırasıyla Düzenle|Seç|Sıra, Düzenle|Seç|Sütun, ve Düzenle|Seç|Hepsini menülerine tekabül ederler. Bizim yazdığımız kod QTable sınıfın selectRow(), selectColumn(), ve selectCells() fonksiyonlarına itimad eder.

```
void Spreadsheet::findNext(const QString &str,bool caseSensitive)
{
      int row = currentRow();
      int col = currentColumn() + 1;
      while (row < NumRows)
      {
            while (col < NumCols)
            {
                  if (text(row, col).contains(str,caseSensitive))
                  {
                        clearSelection();
                        setCurrentCell(row, col);
                        setActiveWindow();
                        return;
                  }
                  ++col;
            }
            col = 0;
            ++row;
      }
      qApp->beep();
}
```

findNext() dilimi imlecin sağındaki hücreden başlayarak cari satırın sonuna varıncaya kadar arar, şayet satırın sonuna varıldıysa bir sonraki satırın başından aramaya devam eder. Bu işlem aranan metin bulununcaya yada en son hücreye varıncaya kadar devam eder. Mesela cari hücre C27 ise, biz D27, E27, &, Z27 hücrelerini, daha sonrada A28, B28, C28, &, Z28, vesaire Z999 hücresine varıncaya kadar arıyoruz. Aranılan text bulundu ise hali hazırdaki seçilmiş alanı iptal edip imleci aradığımız metni içeren hücreye yerlerştiriyoruz ve bu çizelgeyi içeren pencereyi aktif yapıyoruz. Şayet aranılan metin bulunamadı ise bu durumda proğram küçük bir ses (beep) çıkararak aramanın başarısızlıkla sonuçlandığını gösterir.

```
void Spreadsheet::findPrev(const QString &str, bool caseSensitive)

{
      int row = currentRow();
      int col = currentColumn() - 1;
```

```
        while (row >= 0)
        {
            while (col >= 0)
            {
                if (text(row, col).contains(str,caseSensitive))
                {
                    clearSelection();
                    setCurrentCell(row, col);
                    setActiveWindow();
                    return;
                }
                --col;
            }
            col = NumCols - 1;
            --row;
        }
        qApp->beep();
    }
```

findPrev() dilimi findNext() dilimine çok benzer. Aralarındaki tek fark findPrev() fonksiyonunun geriye doğru arayıp A1 hücresinde durmasıdır.

Diğer Menülerin Oluşturulması

Şimdi Aletler ve Seçenekler menüleri için gerekli olan dilimleri oluşturacağız.




**Şekil 4.9**: Spreadsheet proğramının Aletler ve Seöenekler menüleri.

```
    void Spreadsheet::recalculate()
    {
        int row;
        for (row = 0; row < NumRows; ++row)
        {
            for (int col = 0; col < NumCols; ++col)
            {
                if (cell(row, col))
                cell(row, col)->setDirty();
            }
        }
        for (row = 0; row < NumRows; ++row)
        {
            for (int col = 0; col < NumCols; ++col)
            {
                if (cell(row, col))
                updateCell(row, col);
            }
        }
    }
```

recalculate() dilimi Aletler|Yeniden Hesapla menüsüne tekabül eder. Bu dilim Spreadsheet tarafındanda gerektiğinmde çağrılmaktadır. Bütün hücreleri ziyaret edip yeniden hesaplanması gereken her hücre için setDirty() fonksiyonunu çağırarak nişanlıyoruz. Bu aşamadan sonra ne zamanki QTable, text() fonksiyonunu bir hücrenin deşerini elde edip ekranda görüntülemek için çağırırsa o anda hücrenin değeri yeniden hesaplanır. Sonra biz updateCell() fonksiyonunu kullanmak suretiyle bütün elektronik öizelgenin yeniden görüntülenmesini sağlıyoruz. QTable daki yeniden görüntüleme işlemini yapan kod text() fonksiyonunu çağırarak herbir hücrenin değerini görüntülemek üzere elde eder. Daha önce her bir hücre için setDirty() fonksiyonunu çağırmış olmamız hasebiyle text() fonksiyonu en son hesaplanan değerleri dönderir. Hesaplamalar Cell sınıfı tarafından ifa edilir.

```
void Spreadsheet::setAutoRecalculate(bool on)
{
      autoRecalc = on;
      if (autoRecalc)
            recalculate();
}
```

setAutoRecalculate() dilimi Seçenekler|Otomatikmen Yeniden Hesapla menüsüne tekabül eder. Eğer bu seşenek tercik edilmiş ise, elektronik öizelgenin güncelliğini garanti etmnek iöin, biz hemen bütün hücreleri yeniden hesaplıyoruz. Bundan sonra recalculate() fonksiyonu somethingChanged() tarafından bir değişikliği mütaakip otomatik olarak çağrılır.

Seçenekler|Izgarayı Göster menüsü için hususi bir kod yazmamıza gerek yok çünkü QTable zaten setShowGrid(bool) dilimini tedarik etmektedir. Geriye kalan sadece Spreadsheet:: sort() fonksiyonudurki buda MainWindow::sort() tarafından çağrılmaktadır:

```
void Spreadsheet::sort(const SpreadsheetCompare &compare)
{
      vector<QStringList> rows;
      QTableSelection sel = selection();
      int i;
      for (i = 0; i < sel.numRows(); ++i)
      {
            QStringList row;
            for (int j = 0; j < sel.numCols(); ++j)
            row.push_back(formula(sel.topRow() +
                              i, sel.leftCol() + j));
            rows.push_back(row);
      }
      stable_sort(rows.begin(), rows.end(), compare);
      for (i = 0; i < sel.numRows(); ++i)
      {
            for (int j = 0; j < sel.numCols(); ++j){
                  setFormula(sel.topRow() + i,
                        sel.leftCol() + j, rows[i][j]);

            }
      }
      clearSelection();
      somethingChanged();
```

```
}
```

Tasnif yada sıralama işlemi tasnif anahtarlarını kullnarak seçilmiş olan hücreler için gerçekleştirilir. Seçilmiş olan hücrelerin her bir satırını yada sırasını bir QStringList listesinde tutuyoruz ve daha sonra bu satırları bir vektörde (vector<T>) topluyoruz. vector<T> sınıfı standart C++ ın bir parçasıdır; onbirinci bölümde muhtevi sınıfları altında açıklanacaktır. Kolaylık olsun diye değerler yerine formülleri kullanarak sıralamayı gerçekleştiriyoruz.

| | C | D | E |
|---|---|---|---|
| 2 | Edsger | Dijkstra | 1930-05-11 |
| 3 | Tony | Hoare | 1934-01-11 |
| 4 | Niklaus | Wirth | 1934-02-15 |
| 5 | Donald | Knuth | 1938-01-10 |

| index | value |
|---|---|
| 0 | [ "Edsger", "Dijkstra", "1930-05-11" ] |
| 1 | [ "Tony", "Hoare", "1934-01-11" ] |
| 2 | [ "Niklaus", "Wirth", "1934-02-15" ] |
| 3 | [ "Donald", "Knuth", "1938-01-10" ] |

**Şekil 4.10**: Seçilmiş olan hücrelerin sıralar halinde bir vektörde tutulması.

Satırların tasnif işlemini geröekleştirmek için bir standart C++ da yer alan olan stable_sort() fonksiyonunu çağırıyoruz. stable_sort() fonksiyonunun bir başla iteratörüne, bir bitir iteratörüne ve birde karılaştırma fonksiyonuna ihtiyacı vardır. Karşılaştırma yada kıyas vazifesine gören fonksiyon iki tane QStringLists türünde argüman alır ve eğer ilk argüman ikincisinden küçük ise müsbet (true) aksi taktirde menfi (false) değerini üretir. Karşılaştırma fonksiyonu olarak kullandığımız kıyas nesnesi aslında bir fonksiyon değildir andcak, birazdan göreceğimiz gibi, fonksiyonmü gibi kullanılabilir.

| index | value |
|---|---|
| 0 | [ "Donald", "Knuth", "1938-01-10" ] |
| 1 | [ "Edsger", "Dijkstra", "1930-05-11" ] |
| 2 | [ "Niklaus", "Wirth", "1934-02-15" ] |
| 3 | [ "Tony", "Hoare", "1934-01-11" ] |

| | C | D | E |
|---|---|---|---|
| 2 | Donald | Knuth | 1938-01-10 |
| 3 | Edsger | Dijkstra | 1930-05-11 |
| 4 | Niklaus | Wirth | 1934-02-15 |
| 5 | Tony | Hoare | 1934-01-11 |

**Şekil 4.11**: Tasnif işleminden sonra datanın hücrelere geri kopyalanması.

Tasniş işlemini geröekleştirdikten sonra datayı tabloya gerş kopyalıyo, seçilmiş alanı temzliyor ve somethingChanged() fonksiyonun çağırıyoruz. spreadsheet.h dosyasında SpreadsheetCompare sınıfı üu ürkilde tanımlanmıştı:

```
class SpreadsheetCompare
{
public:
      bool operator()(const QStringList &row1,
                  const QStringList &row2) const;
      enum { NumKeys = 3 };
      int keys[NumKeys];
      bool ascending[NumKeys];
};
```

SpreadsheetCompare sınıfı () operatörü tanımlaması itibariyle farklılık arzeder. Bu

operatörün tanımlanmış olması bu sınıfı sanki bir fonksiyonmüş gibi kullanmamızı sağlar. Bu tür sınıflara "functors[12]". Fanktörlerin nasıl çalıştığını anlamak için basit bir misal ile başlayacağız:

```
class Square
{
public:
      int operator()(int x) const { return x * x; }
};
```

Square sınıfının sadece bir fonksiyonu vardır ki oda  operator()(int) operatörüdür. Bu fonksiyon argümanının karesini üretir. Fonksiyona  normal bir isim (mesela ompute(int)) vermey yerine  operator()(int) diye adlandırmış olmamız Square türündki bir nesneyi fonksiyon olarak kullanmamaıza imkan sağlar:

```
Square square;
int y = square(5);
```

Şimdi SpreadsheetCompare sınıfını içeren bir misali ele alalım:

```
QStringList row1, row2;
SpreadsheetCompare compare;
...
if (compare(row1, row2))
{
// row1 is less than row2
}
```

compare nesnesi sanki o normal compare() isminde bir fonksiyonmuş gibi kullanılır.  İlave olarak sınıfın üye değişkenlerinde saklanmış olan tasnif anahtarlarına ve tasnıf sıralamalarına ulaşabilir.

Bu yaklaşıma bir alternatif olarak tasnif anahtarlarını ve sıralarını küresel (global) değişkenlerde saklayıp normal bir  compare() fonksiyonu kullanmak olurduç Ancak küresel değişkenler vasıtası ile haberleşme zarif olmadığı gibi bulması zor lan hata veya çaparlara yol aöabilir. Fanktörs stable_sort() gibi template fonksşyonlarla kullanma hususunda normal fonksiyonlardan daha elverişlidirler.

İşte elektronik çizelge proğramından iki satırı karşılaştıran fonksiyon:

```
bool SpreadsheetCompare::operator()(const QStringList &row1,
            const QStringList &row2) const
{
      for (int i = 0; i < NumKeys; ++i)
      {
            int column = keys[i];
            if (column != -1)
            {
                  if (row1[column] != row2[column])
                  {
                        if (ascending[i])
```

[12] fanktör diye telaffuz edilir

```
                                                return row1[column] < row2[column];
                                          else
                                                return row1[column] > row2[column];
                                    }
                              }
                        }
                        return false;
                  }
```

İlk satırın ikinci satırdan küçük olması durumunda müsbet (true), aksi taktirde menfi (false) değerlerini üretir. Standart stable_sort() fonksiyonu bu fonksiyonun ürettiği değerleri kullanarak rasnif işlemini ifa eder. SpreadsheetCompare nesnesinin keys ve ascending listeleri MainWindow::sort() fonksiyonunda doldurulmaktadırlar (ikinci bölümde gösterildi). Her bir key (anahtar) bir sütun indeksi, yada -1 ("None[13]").

İki satırdaki mütekabil hücre iöeriklerini sırasıyla kıyas ediyoruz. Bir fark bulur bulmaz müsbet yada menfi değerini geri gçnderiyoruz. Eğer bütün karşılaştırmalar eşit ise menfi geri gçnderiyoruz. stable_ sort() fonksiyonu satırların rasnifden önceki durumlarını göz önünde bulundurarak berabere kalma durumlarını çözüyor; row1 (satır1) , row2 (satır2) den önce geldi ise ve herhangi birisi diğerinden küçük değil ise row1 yerini mhafaza eder. std::stable_ sort() fonksşyonunu bundan daha meşhur ancak daha az tutarlı olan std::sort() fonksiyonundan temyiz edende budur.

Böylece Spreadsheet sınıfını tamamlamış olduk. Geleck kısımda Cell sınıfın yeniden gözden geçireceğiz. Bu sınıf hücre formüllerini ihtiva etmekle beraber Spreadsheet tarafından yeniden hesaplanan değerlerin gösterilmesi maksadıyla çağrılan text() fonksiyonun yeniden tanımlanmış halinide içerir.

## *QTableItem dan Altsınınf Oluşturulması*

Cell sınıfı QTableItem ın bir altsınıfıdır. Bu sınıf Spreadsheet sınıfı ile gayet elverişli bir şekilde çalışmaıs için tasarlanmıştır ancak aralarındaki bağ gayet zayıf olduğundan Cell sınıfı her hangi bir QTable ile kullanılabilir. İşte başlık dosyası:

```
#ifndef CELL_H
#define CELL_H
#include <qtable.h>
#include <qvariant.h>
class Cell : public QTableItem
{
public:
      Cell(QTable *table, const QString &formula);

      void setFormula(const QString &formula);
      QString formula() const;
      void setDirty();
      QString text() const;
      int alignment() const;
```

---

[13] hiç

```
private:
      QVariant value() const;
      QVariant evalExpression(const QString &str,int &pos) const;
      QVariant evalTerm(const QString &str, int &pos) const;
      QVariant evalFactor(const QString &str, int &pos) const;

      QString formulaStr;
      mutable QVariant cachedValue;
      mutable bool cacheIsDirty;
};
#endif
```

Cell sınıfı üç hususi değişken ilave etmek suretiyle QTableItem sınıfını temdid etmektedir:

- formulaStr değişkeni hücre formülünü QString olarak muhafaza eder.
- cachedValue değişkeni hücrenin değerini Qvariant olarak “cach[14]” eder.
- cacheIsDirty is true if the cached value isn t up to date.

QVariant çoğu C++ ve Qt data türlerini muhafaza edebilir. Bizim bunu kullanmamızın sebebi bazi hücreleri double türünde bir değer bazılarınında QString türünde bir değer içermeleridir.

cachedValue ve cacheIsDirty değişkenleri bir C++ anahtar sözcüğü olan mutable ile tanımlandılar. Buda bizim bu değişkenleri const fonksiyonları işerisinde değiştirmemize izin verir. Alternatif olarak her text() fonksiyonu çağrıldığında değerleri yeniden hesaplamakta mümkün, ancak bu gereksiz yere verimliliği düşürür.

Dikkat edilirse Q_OBJECT makrosu sınıfın tanımında yer almamaktadır. Cell sinyal ve dilimleri olmayan normal bir C++ sınıfıdıri. Aslında QTableItem,  QObject sınıfının varisi omadığından şu haliyle Cell sınıfında sinyal ve dilimlerin yer alması mümkün değildir. Verimlilği düşürmemek için Qt nin efrad (items) sınıfları QObject sınıfının varısleri yapılmamışlardır. Sinyal ve dilimlerin gerekmesi durumunda onlar efradı ihtiva eden alet iöerisinde tanımlanabilirler yada nadiren müteaddid veraset kullanarak QObject sınıfının alt sınıfı yapılabilirler. İşte cell.cpp dosyasının baş kıasmı:

```
#include <qlineedit.h>
#include <qregexp.h>
#include "cell.h"
Cell::Cell(QTable *table, const QString &formula)
      : QTableItem(table, OnTyping)
{
      setFormula(formula);
}
```

Yapıcı (constructor ) QTable ın bir müşirini ve bir formül gerektirmektedir. Müşir QTableItem yapıcısına gönderilir ve daha sonra bu müşire QtableItem::table() vasıtası ile

[14] keş diye telafuz edilir

ulaşılabilir. Temel sınıfın (`QtableItem`) ikinci argümanı olan OnTyping[15] ise muharririn (editör) kullanıcı aktif hücreye yazmasına başlar başlamaz gözükmesi anlamına gelir.

```
void Cell::setFormula(const QString &formula)
{
      formulaStr = formula;
      cacheIsDirty = true;
}
```

setFormula() fonksiyonu hücrenin formülünü tesbit eder. Aynı zamanda cacheIsDirty değişkeninin değerinide müsbet (true) yapar ki buda cachedValue değerinin kullanılmadan evvel yeniden hesağlanması gerektiğini ifade eder. Bu fonksiyon Cell sınıdının yapıcısı ve Spreadsheet:: setFormula() fonksiyonları tarafından çağrılır.

```
QString Cell::formula() const
{
      return formulaStr;
}
```

formula() fonksiyonu, Spreadsheet::formula() tarafından çağrılır.

```
void Cell::setDirty()
{
      cacheIsDirty = true;
}
```

setDirty() fonksiyonu hücrenin değerinin yeniden hesaplanmasını zorunlu kılmak maksadıyla çağrılır. Bu fonksiyonun yaptığı tek şey cacheIsDirty değişkeninin değerini müsbet (true) olarak deüiştirmektir. Değerin yeniden hesaplanması işlemi ne zaman gerekli ise o zaman gerçekleştirilir.

```
QString Cell::text() const
{
      if (value().isValid())
            return value().toString();
      else return "####";
}
```

text() fonksiyonu QtableItem dan miras kalmakla birlikte burada yeniden tanımlanmaktadır. Elektronik öizelgede görüntülenmesi gereken metni getirir. Hücrenin değerinin hesaplanmasında value() fonksiyonuna itimad etmektedir. Şayet hücrenin değeri muteber değil ise yani geçersiz ise (bu genelde formülün yanlış olmasından kaynaklanır) bu durumda "####" metnini geri gönderiyoruz .

text() fonksiyonu tarafından kullnılan value() fonksiyonu QVariant türünde bir dğişken üretir. QVariant, double ve QString gibi değişik data türlerini muhafaza edebilir ve üye fonksiyonları vasıtasıyla variant diğer data türlerine çevrilebilirler. Mesela, double türünde bir değer içeren varıand içim toString() fonksiyonunu çağırmakla double, metin olarak temsil edilebilir. Varsayılan (default) yapıcıyı kullanarak husule getirilen QVariant muteber yada geöerli bir variant değildir.

---

[15] yazmaya başlar başlamaz

```
int Cell::alignment() const
{
      if (value().type() == QVariant::String)
            return AlignLeft | AlignVCenter;
      else
            return AlignRight | AlignVCenter;
}
```

alignment[16]() fonksiyonu aslen QtableItem yer almakla birlikte biz burada onu yeniden tanımladık. Bu fonksiyon bir hücrenin metni için hiza ayarını dönderir. Biz matinleri sola yanaşık ve rakamları sağa yanaşık olarak hizalamayı tercih ettik. Düşey doğrultudaki hizalamaya gelince bütün değerler merkezi olarak hizalanırlar.

```
const QVariant Invalid;
QVariant Cell::value() const
{
      if (cacheIsDirty)
      {
            cacheIsDirty = false;
            if (formulaStr.startsWith(" "))
            {
                  cachedValue = formulaStr.mid(1);
            }
            else if (formulaStr.startsWith("="))
            {
                  cachedValue = Invalid;
                  QString expr = formulaStr.mid(1);
                  expr.replace(" ", "");
                  int pos = 0;
                  cachedValue = evalExpression(expr, pos);
                  if (pos < (int)expr.length())
                  cachedValue = Invalid;
            }
            else
            {
                  bool ok;
                  double d = formulaStr.toDouble(&ok);
                  if (ok)
                        cachedValue = d;
                  else
                        cachedValue = formulaStr;
            }
      }
      return cachedValue;
}
```

Hususi value() fonksiyonu hücrenin değerini dönderir. Eğer cacheIsDirty değişkeninin değeri müsbet ise hücrenin değeri yeniden hesaplanmalıdır.

Eğer formül tek tırnak işareti ile başlarsa (mesela, “'12345”), hücrenin değeri metnin ikinci indexinden sonuna kadar olan değerdir. (Tek tırnak ilk indekste yer almaktadır)

[16] hiza

Eğer formül "=" ile başlarsa metnin ikinci indexden başlayarak varsa bütün boşlukları siliyoruz. Daha sonra evalExpression() fonksiyonunu çağırarak formülü değerlendiriyoruz.

pos argümanı "passed by reference" türünde bir argümandır ve tahlilin (parsing) başlayacağı indexi gösterir. evalExpression() çağrıldıktan sonra pos deüişkeni başarılı bir şekilde tahlil edilebilen mentnin uzunluğuna eşitlenir. Eğer tahlil işlemi, metnin sonuna ermeden, başarısızlıkla sonuçlandı ise cachedValue deüişkeninin değerini Invalid (geşersiz) olarak değiştiriypruz.

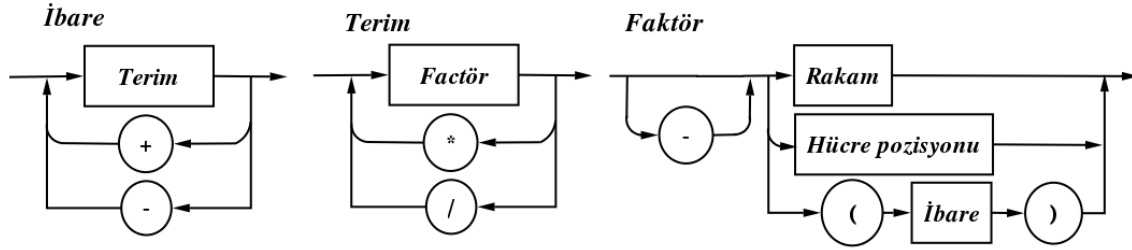
Eğer formül tek tırnak işareti veya eşitlik işareti ('=') ile başlamıyor ise toDouble() fonksiyonunu çağırarak onu real sayıya çevirmeye cehd ediyoruz. Eğer çeviri başarılı ise cachedValue değişkenine budeğeri yerleştiriyoruz; aksi takdirde cachedValue değişkenine meni aynen yerleştiriyoruz. Mesela, formülün 1.50 olduğunu varsayarsak ve bunu toDouble() fonksiyonu ile doğal sayıya çevirmeye kalktığımızda elde edeceğimiz değer 1.5 dir, binaenaleyh "Dünaynın Nüfusu" formülünü toDouble() fonksiyonu ile doğal sayıya çevirmeye kalkarsak `ok` değişkeni menfiye ayarlanmış olur ve elde edeceğimiz değer 0.0 dır.

value() fonksiyonu "const" bir fonksiyondur. Bu yüzden cachedValue ve cacheIsValid deişkenlerini mutable olarak tanımladıkk taki derleyici bu değişkenleri const bir fonksiyon içerisinde değiştimemize izin versin. value() fonksiyonununu const fonksiyon yerine normal fonksiyon yapıp mutable anahtar sözcüğünü kullanmamayı düşünebilirsiniz ancak value() fonksiyonu const bir fonksiyon olan text() tarafından çağrıldığından onu derleyemezsiniz. C++ diline caching ve mutable genelde beraber kullanılırlar.

Formüllerin tahlili haricinde Spreadsheet proğramını böylece tamalamış olduk. Bu bölümün geri kalan kısmı evalExpression() fonksiyonu ile evalTerm() ve evalFactor() yardımcı fonksiyonlarını içine almaktadır. Kode biraz karmaşık ancak proğramın tamam olması iöin buraya dahil edildi. Bu kodun GUI ile ilgisi olmadığı içim arzu ederseniz bu kısmı atlayıp beşinci bölümden devam edebilirsiniz.

evalExpression()[17] fonksiyonu elektronik çizelge ibarelerini değerlendirir. İbare birbirlerin toplama ('+') veya çıkarma ('-') işaretleri ile ayrılmış terimler topluluğu olarak tanımlanır; mesela, "2*C5+D6" bir ibare olup, "2*C5" ibarenin ilk terimi ve D6 ise ikinci terimidir. Terimler ise yine birbirlerinden çarpma ('*') veya bölme ('/') işaretleri ile ayrılmış faktörler topluluğu olarak tanımlanır; mesela, "2*C5", ilk faktörü "2" ve ikinci faktörü "C5" olan bir terimdir. Faktör ise bir sayı ("2"), bir hücre pozisyonu ("C5"), veya önünde muhtemelen bir eksi (-) işareti bulunan parantez içine alınmış bir ibare olabilir. İbareleri terimlere ve terimleri faktörlere ayırmak suretiyle operatörlerin (çarpma, toplama gibi) öncelik sırasına göre değerlendirilmelerini garanti altına almış oluyoruz.

[17] ibareyi değerlendir




**Şekil 4.12**: Elektronik çizelge proğramının nahiv (sintaks) diyagramı.

Elektronik çizelge ibarelerinin nahvi şekil 4.12 tasvir edilmektedir. Gramerdeki her bir sembola mğkabil (İbare, Terim, ve Faktör), Cell sınıfının bir üye fonksiyonu mevcuttur ki nu fonksiyon mukabili olan sembolü tahlil eder ve yapı itibarı ile sembolün yapısına benzer. Bu şekilde yazılmış yada tasarlanmış olan tahlil edicilere nüzulu mütedahil (recursive-descent) adı verilir. İbareleri tahkik eden evalExpression() fonksiyonu ile başlayalım:

```
QVariant Cell::evalExpression(const QString &str, int &pos) const
{
      QVariant result = evalTerm(str, pos);
      while (pos < (int)str.length())
      {
            QChar op = str[pos];
            if (op !=  '+'  && op !=  '-' )
                  return result;
            ++pos;

            QVariant term = evalTerm(str, pos);
            if (result.type() == QVariant::Double
                  && term.type() == QVariant::Double)
            {
                  if (op ==  '+' )
                        result=result.toDouble()+term.toDouble();
                  else
                        result=result.toDouble()-term.toDouble();
            }
            else
            {
                  result = Invalid;
            }
      }
      return result;
}
```

Önce ,evalTerm() fonksiyonunu çağırıp ilk terimi elde ediyoruz. Eğer müteakip karekter bir '+' veya '-' ise evalTerm() fonksiyonunu ikinci defa çağırarak devam ediyoruz; aksi takdirde ibare bir terimden ibarettir ve bu terimi değerlendirip onu ibarenin değeri olarak geri gönderiyoruz. İlk iki terimin değerini hesapladıktan sonra aralarındaki operatörü (çarpa, toplama vesaire) kullanarak sonucu hesap ediyoruz.  Eğer terimlerin her ikisininde değeri doğal sayı ise bu durumda sonucu double olarak hesaplıyoruz, aksi halde sonucu Invalid (geçersiz) olarak belirliyoruz. Bu şekilde bütün terimleri bitirinceye kadar devam ediyoruz.

Bunun doğru olarak öalışmasının sebebi toplama ve şıkarma işlemkerinin soldan sağa doğru (left-associative) yapılmalarıdır; yani, "1+2+3" ibaresi "1+(2+3)" olarak değil bilakis "(1+2)+3" olarak algılanır ve değerlendirilir.

```
QVariant Cell::evalTerm(const QString &str, int &pos) const
{
    QVariant result = evalFactor(str, pos);
    while (pos < (int)str.length())
    {
         QChar op = str[pos];
         if (op !=  '*'  && op !=  '/' )
            return result;
         ++pos;
         QVariant factor = evalFactor(str, pos);
         if (result.type() == QVariant::Double
            && factor.type() == QVariant::Double)
         {
            if (op ==  '*' )
            {
              result=result.toDouble()*factor.toDouble();
            }
            else
            {
              if (factor.toDouble() == 0.0)
                  result = Invalid;
              else
                  result = result.toDouble() / factor.toDouble();
            }
         }
         else
         {
            result = Invalid;
         }
    }
    return result;
}
```

evalTerm()[18] fonksiyonu evalExpression() fonksiyonuna çok benzemekle birlikte toplama ve çıkarma ilemleri yerine çarpama ve bölme işlemleri ile iştigale etmektedir. evalTerm() fonksiyonunda göz önünde bulundurulması gereken tek husus sıfıra bölme işleminden kaçınmaktır. Genelde yuvarlama hataları yüzünden oating point (kayan noktalı) değerlerin eşitlik testi tavsiye edilmez ancak sıfıra bölme işlemindan kaşınmak için bunu yapmakta bir sakınca yoktur.

```
QVariant Cell::evalFactor(const QString &str, int &pos) const
{
      QVariant result; bool negative = false;
      if (str[pos] ==  '-' )
      {
            negative = true; ++pos;
```

[18] terimi değerlendir, terimin değerini hesapla

```
        }
        if (str[pos] ==  '(' )
        {
              ++pos;
              result = evalExpression(str, pos);
              if (str[pos] !=  ')' )
                    result = Invalid;
              ++pos;
        }
        else
        {
              QRegExp regExp("[A-Za-z][1-9][0-9]{0,2}");
              QString token;
              while (str[pos].isLetterOrNumber()|| str[pos]== '.')
              {
                    token += str[pos]; ++pos;
              }
              if (regExp.exactMatch(token))
              {
                    int col = token[0].upper().unicode() - 'A' ;
                    int row = token.mid(1).toInt() - 1;
                    Cell *c = (Cell *)table()->item(row, col);
                    if (c)
                          result = c->value();
                    else
                          result = 0.0;
              }
              else {
                    bool ok;
                    result = token.toDouble(&ok);
                    if (!ok)
                          result = Invalid;
              }
        }
        if (negative)
        {
              if (result.type() == QVariant::Double)
                    result = -result.toDouble();
              else
                    result = Invalid;
        }
        return result;
  }
```

evalFactor()[19] fonkisyonu evalExpression() ve evalTerm() fonksiyonlarından biraz dahah karmaşıktır. Önce faktürün önünde eksi işareti olup olmadığını tesbit etmekle işe başlıyoruz. Daha sonra faktörün bir açık parantez ile başlayıp başlamadığına bakıyoruz. Eşer faktör açık parantez ile başlıyor ise evalExpression() fonksiyonunu öağırmak suretiyle parantez iöerisindeki ibareyi değerlendiriyoruz. İşte tahlil edicinin (parser) bu aşamasında tedahül (recursion) veya tedai diye tabir edilen makanizma yer alır; evalExpression() fonksiyonu, evalTerm() fonksiyonunu, oda evalFactor() fonksiyonunu, buda tekrar evalExpression()

[19] faktörü değerlendir

fonksiyonunu şağırır.

Eğer faktör mütedahil (nested) ibare değil ise, faktörün bir sonraki alt birimini çıkarıyoruz. Eğer bu alt birim QRegExp kriterine uygun ise, onu hücre pozisyonu olarak algılıyor ve bu pozisyondaki hücrenin değerini value() fonksiyonunu kullanarak alde ediyoruz. Hücre elektronik çizelgenin herhangi bir noktasında yer alabileceği gibi başka hücrelerede bağımlı olabilir. Hücrelerin yekdiğerine bağlılıkları bir problem teşkil etmez; her bir baüımlılık için value() fonksiyonu çağrılır ve "kirli" yani yeniden hesaplanması gereken hücreler tahlil edilir taki bütün bağımlılıklar halledilsin. Eğer faktörün alt birimi hücre pozisyonu değil ise onu sayı olarak algılıyoruz.

A1 hücresinin "=A1" formülünü ihtiva etmesi durumunda ne olur? Yada A1 hücresi "=A2" formülünü ve A2 hücreside "=A1" formülünü içeriyor ise nasıl bir sonuç elde edilir? Her ne kadar kısır döngüleri tahlil etmek için hususi bir kod yazmadıysakta, hali hazırdaki kod böyle durumların farkına varıp geçersiz bir Qvariant döndermektedir. evalExpression() fonksiyonunu çağırmadan önce value() fonksiyonu içerisinde cacheIsDirty değişkenini menfi ve cachedValue değişkenini geçersiz (Invalid) yaptığımızdan dolayı bir problem iıkmamaktadır. Eğer evalExpression() fonksiyonu tedai (recursively) bir şekilde aynı hücre için çağırırsa, anında value() fonksiyonu "Invalid" değerini geri gönderir ve ibarenin tamamının değeri geçersiz olarak belirlenir.

Bölece elektronik çizelge proğramının tahlil edicisini tamamlamış olduk. Bu aşamada artık tahlil edicinin, sum() ve avg() gibi standart elektronik çizelge fonksiyonlarını anlayabilmesi için ilaveler (faktörün gramer tanımını genişleterek) yapmak kolay olsa gerektir.  Bir diğer kolay ilavede "+" operatürünün sadece sayıları değil aynı zamanda metinleride toplama (yada ekleme) yeteneği vermektir. Bu gramerde her hangi bir değişikliği gerektirmez.